Assignment 6.1

2303A51234

Batch 18

Task Description #1 (AI-Based Code Completion for Loops)

Task: Use an AI code completion tool to generate a loop-based

program.

Prompt:

"Generate Python code to print all even numbers between 1 and N

using a loop."

Expected Output:

• AI-generated loop logic.

• Identification of loop type used (for or while).

• Validation with sample inputs.

#Prompt:

#Generate Python code to print all even numbers between 1 and N

#using loop

#Identification of loop type used (for or while)

#code:

```python
N = int(input("Enter a number N: "))
print("Even numbers between 1 and", N, "are:")
for num in range(1, N + 1):
    if num % 2 == 0:
        print(num)
```

#output:

```
Enter a number N: 7
Even numbers between 1 and 7 are:
2
4
6
PS C:\Users\aksha\OneDrive\Desktop\devops> 
```

#analysis:

1.AI generated correct code.

2.Output was accurate.

3. Easy to use in vs.

Task Description #2 (AI-Based Code Completion for Loop with Conditionals)

Task: Use an AI code completion tool to combine loops and

conditionals.

Prompt:

"Generate Python code to count how many numbers in a list are even

and odd."

Expected Output:

• AI-generated code using loop and if condition.

• Correct count validation.

• Explanation of logic flow.

#prompt:
#Generate Python code to count how many numbers in a list are even and odd.

#Correct count validation

#Explanation of logic flow

#code:
```python
numbers = [10, 15, 22, 33, 42, 55, 60]
even_count = 0
odd_count = 0
for number in numbers:
    if number % 2 == 0:
        even_count += 1
    else:
        odd_count += 1
print("Even numbers count:", even_count)
print("Odd numbers count:", odd_count)
```

#output:
```
\.vscode\extensions\ms-python.debugpy-2025.18.0-w1
Even numbers count: 4
Odd numbers count: 3
```

#analysis:

• One example helped the AI generate correct code.

• The task improved understanding of AI pattern learning.

• Clear prompts gave accurate output.

Task Description #3 (AI-Based Code Completion for Class Attributes Validation)
Task: Use an AI tool to complete a Python class that validates user

input.

Prompt:

"Generate a Python class User that validates age and email using

conditional statements."

Expected Output:

• AI-generated class with validation logic.

• Verification of condition handling.

• Test cases for valid and invalid inputs.

#prompt:
#Generate a Python class User that validates age and email using conditional statements.

#code:

```python
class User:
    def __init__(self, name, age, email):
        self.name = name
        self.age = age
        self.email = email
        self.validate_age()
        self.validate_email()

    def validate_age(self):
        if self.age < 0 or self.age > 120:
            raise ValueError("Age must be between 0 and 120.")

    def validate_email(self):                    (parameter) self: Self@User
        if "@" not in self.email or "." not in self.email.split("@")[-1]:
            raise ValueError("Invalid email format.")
```

```python
# Test cases
user1 = User(20, "test@example.com")
print(user1.validate_age())
print(user1.validate_email())

user2 = User(15, "testexample.com")
print(user2.validate_age())
print(user2.validate_email())
```

#output:

```
\.vscode\extensions\ms-python.debugpy-2023.18.0-win32-x64\bundled\libs\debugpy\la
Valid age
Valid email
Invalid age
Invalid email
```

#analysis:

- examples helped the AI understand name formatting.

- The task improved accuracy in code generation.

- Clear prompts produced correct output.

Task Description #4 (AI-Based Code Completion for Classes)

Task: Use an AI code completion tool to generate a Python class for

managing student details.

Prompt:

"Generate a Python class Student with attributes (name, roll number,

marks) and methods to calculate total and average marks."

Expected Output:

• AI-generated class code.

• Verification of correctness and completeness of class

structure.

• Minor manual improvements (if needed) with justification.

Prompt:

#Generate a Python class Student with attributes (name, roll number,marks) and methods to calculate total and average marks.

#code:

```python
class Student:
    def __init__(self, name, roll_number, marks):
        self.name = name
        self.roll_number = roll_number
        self.marks = marks   # marks should be a list of integers

    def calculate_total(self):
        return sum(self.marks)

    def calculate_average(self):
        if len(self.marks) == 0:
            return 0
        return self.calculate_total() / len(self.marks)
```

```python
student1 = Student("Alice", 101, [85, 90, 78, 92])
print("Total marks:", student1.calculate_total())
print("Average marks:", student1.calculate_average())
student2 = Student("Bob", 102, [70, 75, 80])
print("Total marks:", student2.calculate_total())
print("Average marks:", student2.calculate_average())
student3 = Student("Charlie", 103, [])
print("Total marks:", student3.calculate_total())
print("Average marks:", student3.calculate_average())
```

#output:

```
Total marks: 345
Average marks: 86.25
Total marks: 225
Average marks: 75.0
Total marks: 0
Average marks: 0
```

#analysis:

- One example helped the AI generate correct code.

- The task improved understanding of AI pattern learning.

- Clear prompts gave accurate output.

Task Description 5 (AI-Assisted Code Completion Review)

Task: Use an AI tool to generate a complete Python program using

classes, loops, and conditionals together.

Prompt:

"Generate a Python program for a simple bank account system using

class, loops, and conditional statements."

#prompt:

#Generate a Python program for a simple bank account system using class, loops, and conditional statements.

#code:

```python
class BankAccount:
    def __init__(self, account_holder, balance=0):
        self.account_holder = account_holder
        self.balance = balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount                    (parameter) self: Self@BankAccount
            print(f"Deposited: {amount}. New balance: {self.balance}")
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        if amount > 0:
            if amount <= self.balance:
                self.balance -= amount
                print(f"Withdrew: {amount}. New balance: {self.balance}")
            else:
                print("Insufficient funds.")
        else:
            print("Withdrawal amount must be positive.")

    def display_balance(self):
        print(f"Account holder: {self.account_holder}, Balance: {self.balance}")
```

```
account = BankAccount("John Doe", 1000)
while True:
    print("\nOptions: 1. Deposit 2. Withdraw 3. Display Balance 4. Exit")
    choice = input("Enter your choice (1-4): ")
    if choice == '1':
        amount = float(input("Enter amount to deposit: "))
        account.deposit(amount)
    elif choice == '2':
        amount = float(input("Enter amount to withdraw: "))
        account.withdraw(amount)
    elif choice == '3':
        account.display_balance()
    elif choice == '4':
        print("Exiting the program.")
        break
    else:
        print("Invalid choice. Please try again.")
```

#output:

```
Options: 1. Deposit 2. Withdraw 3. Display Balance 4. Exit
Enter your choice (1-4): 3
Account holder: John Doe, Balance: 1000
```

#analysis:

1.AI generated correct code.

2.Output was accurate.

3. Easy to use in vs.